Analysis of Algorithms

Algorithms

Words „**algorithm**“ and „**algebra**“ - Abu Ja'far Mohammed ibn Mûsâ al-Khowârizmî – ca 825 AD, rules for performing arithmetic operations

Algorithm is an unambiguous (exact) instruction for solving a given task.

Algorithm consists of finite number of steps, each single step takes the final amount of time and final amount of other resources. Even so, an algorithm may not halt (terminate)!

Algorithm may have inputs (set of input data) and outputs (set of output data) - algorithm is often considered to be a data processing function.

If an algorithm finishes its work (terminates) for all possible inputs, it is called **total**.

If an algorithm may not terminate for some inputs, it is called **partial**.

If after each step the next step is uniquely defined, the algorithm is called **deterministic**, otherwise **non-deterministic**. Non-deterministic algorithm may give different outputs when executed on the same inputs.

Properties of Algorithms

Correctness (narrow) - algorithm meets the specification, solves the „right“ task.

Correctness (wide) - algorithm is correct and safe (e.g. has „reasonable behaviour“ for incorrect or undefined inputs).

Algorithm is **well-defined**, if all the steps are final and unambiguous. Description of an algorithm is always final – possibility to use algorithms as data (John von Neumann).

Halting property - total (always halts, **solvable** tasks) vs. partial (may not halt on some inputs, **semi-solvable** tasks).

Determinism - determined vs. non-determined

Universality - algorithm solves a class of problems, not only some single testcases.

Complexity - time complexity, memory (space) complexity. Average, worst case.

Formal Models of Algorithms

- Turing machine, 1936-37
- Lambda-calculus (Church), 1941
- Post systems, 1943
- Markov algorithms, 1951
- Chomsky 0-type grammars, 1959
- Programming languages, Sammet, 1969

...

Sammet (1969) - all these formal models express the same class of algorithms

Asymptotic Behaviour of Algorithms

n - size of input data

- Time complexity (average A(n), worst case W(n), best case)

f(n) > 0 - running time of an algorithm on input of size n

- Space complexity (average, worst case, best case)

f(n) > 0 - number of memory units needed to run an algorithm on input of size n

Direct measurement using implementation of an algorithm - not always reasonable

Estimation of growth counting „meaningful" operations (f(n) is a number of operations performed by an algorithm on input of size n)

Big-Oh

Given functions $f(n)$ and $g(n)$, we say that $f(n)$ is $O(g(n))$ if there are positive constants c and n_0 such that $f(n) \leq cg(n)$ for all $n \geq n_0$

The big-Oh notation gives an upper bound on the growth rate of a function

The statement "$f(n)$ is $O(g(n))$" means that the growth rate of $f(n)$ is no more than the growth rate of $g(n)$

We can use the big-Oh notation to rank functions according to their growth rate

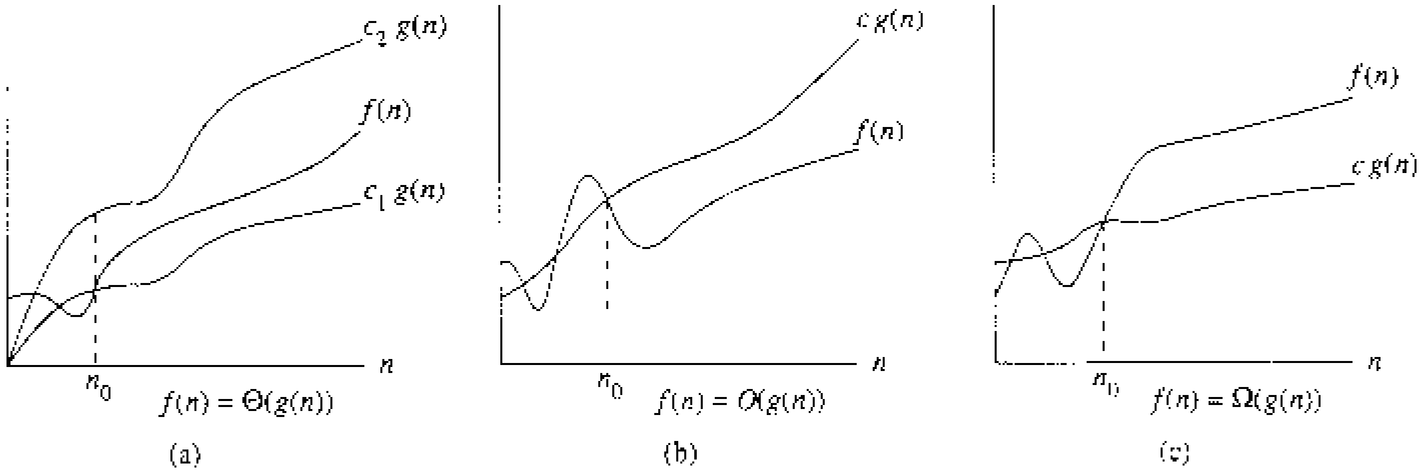

Figure 2.1 Graphic examples of the Θ, O, and Ω notations. In each part, the value of n_0 shown is the minimum possible value; any greater value would also work. **(a)** Θ-notation bounds a function to within constant factors. We write $f(n) = \Theta(g(n))$ if there exist positive constants n_0, c_1, and c_2 such that to the right of n_0, the value of $f(n)$ always lies between $c_1 g(n)$ and $c_2 g(n)$ inclusive. **(b)** O-notation gives an upper bound for a function to within a constant factor. We write $f(n) = O(g(n))$ if there are positive constants n_0 and c such that to the right of n_0, the value of $f(n)$ always lies on or below $cg(n)$. **(c)** Ω-notation gives a lower bound for a function to within a constant factor. We write $f(n) = \Omega(g(n))$ if there are positive constants n_0 and c such that to the right of n_0, the value of $f(n)$ always lies on or above $cg(n)$.

Big-Oh, Big-Omega, Big-Theta

f ~ O(g) <=> $\exists c > 0, \exists \boldsymbol{n_0} > 0, \forall n > \boldsymbol{n_0}$: f(n)/g(n) < c

„f does not grow faster than g“

f ~ Ω(g) <=> g ~ O(f) <=> $\exists b > 0, \exists \boldsymbol{n_1} > 0, \forall n > \boldsymbol{n_1}$: f(n)/g(n) > b

„f does not grow slower than g“

f ~ Θ(g) <=> f ~ O(g) & f ~ Ω(g) <=>

<=> $\forall n > \max(\boldsymbol{n_0}, \boldsymbol{n_1})$: b < f(n)/g(n) < c

„f grows as fast as g“

Little-oh, little-omega

f ~ o(g) <=> $\forall c > 0$: $\exists \boldsymbol{n_0} > 0$: $\forall n > \boldsymbol{n_0}$: $\frac{f(n)}{g(n)} < c$

„f grows (much) slower than g"

f ~ ω(g) <=> g ~ o(f) <=> $\forall b > 0$: $\exists \boldsymbol{n_1} > 0$: $\forall n > \boldsymbol{n_1}$: $\frac{f(n)}{g(n)} > b$

„f grows (much) faster than g"

Asymptotic features of relations

for each constant $a>0$: $f(n) \sim O(af(n))$

if $f(n)<g(n)$ and $g(n) \sim O(h(n))$, then $f(n) \sim O(h(n))$

if $f(n) \sim O(g(n))$ and $g(n) \sim O(h(n))$, then $f(n) \sim O(h(n))$

$f(n)+g(n) \sim O(\max\{f(n), g(n)\})$

if $g(n) \sim O(h(n))$, then $f(n)+g(n) \sim O(f(n)+h(n))$

if $g(n) \sim O(h(n))$, then $f(n)g(n) \sim O(f(n)h(n))$

if $f(n)=p_0+p_1n+...+p_kn^k$ is a polynomial of degree k, then $f(n) \sim O(n^k)$

for each natural number k: $n^k \sim o(2^n)$

for each natural number k: $\log n^k = k \log n \sim O(\log n)$

all logarithms are the same: $\log_b n = \log_a n / \log_a b$ and $a^{\log_a n} = n$

Classes of complexity and examples

O(1) - searching the hash table

O(log n) - binary search

O(sqrt(n)) – function inversion in quantum computing (Grover)

O(n) - „common sense“, naive pattern matching, special sort, ...

O(nlogn) - fast sort with comparision

$O(n^2)$ - naive sort, matrices

$O(n^2 \log n)$

$O(n^3)$ - Floyd-Warshall

... , $O(n^k)$; $O(2^n)$, $O(n!)$, $O(n^n)$, O(2^(2^(2^(...)))) n times, where ^ denotes exponent, ...

Infinite hierarhy of complexities

Ackermann function:

A(m,n)=A(m-1, A(m,n-1)), if m>0 and n>0
A(0,n) = n+1
A(m,0) = A(m-1,1), if m>0

Rekursioon

Keerulisem näide - Ackermanni funktsioon:

A(0, n) = n+1
A(m, 0) = A(m-1, 1)   m>0
A(m, n) = A(m-1, A(m, n-1))   m>0, n>0

	0	1	2	3	4	5
0	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	5	7	9	11	13
3	5	13	29	61	125	253
4	13	65533…				

A(2,1) = A(1,A(2,0)) = A(1,A(1,1)) = A(1, A(0,A(1,0))) =
= A(1, A(0,A(0,1))) = A(1, A(0,1)+1) = A(1,1+1+1) = A(1,3)=
= A(0,A(1,2)) = A(1,2)+1 = A(0,A(1,1))+1 = A(1,1)+1+1 =
= A(0,A(1,0))+2 = A(1,0)+1+2 = A(0,1)+3 = 1+1+3 = 5

Undecidable and hard problems

Halting problem is undecidable

Input: any algorithm in its finite representation and input data for that algorithm

Output: does the algorithm halt when executed on this data (yes/no)?

There is no general algorithm to solve this problem

Hard problem - no polynomial algorithm for the problem is known

Example: Hamiltonian cycle in a graph

$$f \sim O(g) : \exists n_1 \in \mathbb{N}, \exists c_1 \in \mathbb{R} :$$

$$\forall n > n_1 : f(n) \leq c_1 g(n)$$

$$g \sim O(h) : \exists n_2 \in \mathbb{N}, \exists c_2 \in \mathbb{R} :$$

$$\forall n > n_2 : g(n) \leq c_2 h(n)$$

$$n_3 := \max(n_1, n_2)$$

$$\forall n > n_3 : f(n) \leq c_1 g(n) \leq$$

$$\leq c_1 c_2 h(n)$$

$$c_3 := c_1 \cdot c_2$$

$$\exists n_3 \in \mathbb{N}, \exists c_3 \in \mathbb{R} :$$

$$\forall n > n_3 : f(n) \leq c_3 h(n)$$

$$f \sim O(h) \quad \blacksquare$$

Relative growth of running time

	Programmi töö aeg $c\,f(n)$	*Lahendamisaja suhteline suurenemine $f(25)/f(5)$*
	c_1	1
	$c_2 \log n$	2
	$c_3\,n$	5
	$c_4\,n \log n$	10
	$c_5\,n^2$	25
	$c_6\,n^3$	125
	$c_7\,2^n$	1 048 576

Joonis 2.1. Lahendamisaja suhteline kasvamine, kui algandmete maht suureneb 5-lt 25-le.

Keerukus (mikrosek.)	*Suurim ülesanne, mille lahendamise aeg < 1 sek.*	*Suurim ülesanne, mille lahendamise aeg < 1 päev*	*Suurim ülesanne, mille lahendamise aeg < 1 aasta*
n	$n = 1\,000\,000$	$n = 86\,400\,000\,000$	$n = 31\,530\,000\,000\,000$
$n \log_2 n$	$n = 62\,746$	$n = 2\,755\,147\,514$	$n = 798\,160\,978\,500$
n^2	$n = 1000$	$n = 293\,938$	$n = 5\,615\,692$
n^3	$n = 100$	$n = 4\,421$	$n = 31\,593$
2^n	$n = 19$	$n = 36$	$n = 44$
$n!$	$n = 9$	$n = 14$	$n = 16$

Joonis 2.2. Erineva keerukusega programmide ajalised piirid.